

# Std. lib. Functions – Let

```
fun <T, R> T.let(f: (T) -> R): R = f(this)
```

```
DbConnection.getConnection().let { connection ->
}
// connection is no longer visible here
```

You in combination with the null-check operator:

```
val map : Map<String, String> = ...
val config = map[key]
config?.let {
    // This block will not be executed if „config“ is null
}
```

# Std. lib. Functions – Apply

```
fun <T> T.apply(f: T.() -> Unit): T { f(); return this }
```

With apply() we can substitute the “Builder” pattern or simply make our code more readable.

```
val recyclerView: RecyclerView = RecyclerView().apply{  
    setHasFixedSize = true  
    layoutManager = LinearLayoutManager(context)  
    adapter = MyAdapter(context)  
    clearOnScrollListener()  
}
```

# Std. lib. Functions – Run

```
fun <T, R> T.run(f: T.() -> R): R = f()
```

Run() should only be used with lambdas which do not return any values , but only generate sideeffects.

```
webView.settings?.run {  
    javascriptEnabled = true  
    databaseEnabled = true  
}
```

# Std. lib. Functions – Also

```
fun <T> T.also(block: (T) -> Unit): T
```

With **also** you say „also do this with the object“.

Also() passes the object as parameter and returns the same object (not the result of the lambda).

```
val person = Person().also {  
    it.name = "Tony Stark"  
    it.age = 42  
}
```

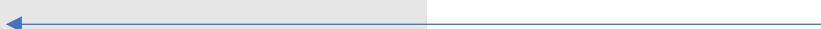
# Std. lib. Functions – With

```
fun <T, R> with(receiver: T, f: T.() -> R): R = receiver.f()
```

It just helps eliminating the repetitive code for setting properties.

By default with() return the result of the last line, if you want to return the object you need to add **this** as the last line.

```
val person = with(Person()){  
    this.age = 42  
    this.name = "Tony Stark"  
    this  
}
```



*Note **with()** doesn't work with nullable variables*

# Std. lib. Functions – Use

```
fun <T : Closeable, R> T.use(block: (T) -> R): R
```

Use function is the equivalent of Java's try-with-resources. It applies to all types of closable instances. It automatically closes the resource (receiver) on exit.

Java style:

```
try(FileReader reader = new FileReader("Input.txt")){  
    // Read file  
}catch (IOException e){  
}  
//automatically closed
```

Kotlin style:

```
FileReader("Input.txt").use {  
    // Read File  
}  
  
// Automatically closed
```

# Std. lib. Functions – takelf

```
fun <T> T.takelf(predicate: (T) -> Boolean): T? = if (predicate(this)) this else null
```

Takelf is a filter for a single value, in combination with the Elvis Operator (?:) you can handle the else case

```
val name: String = "Chris"  
val index = name.indexOf("C").takelf { it > 0 } ?: 0
```

# Std. lib. Functions – takeUnless

```
fun <T> T.takeUnless(predicate: (T) -> Boolean): T? = if (!predicate(this)) this else null
```

TakeUnless is the exact opposite to takeIf(). It takes an inverted predicate.

```
val name: String = "Chris"  
val index = name.indexOf("C").takeUnless { it < 0 } ?: 0
```



# Std. lib. Functions – When to use what

- **Also** : Additional processing on an object in a call chain
- **Apply** : Post-construction configuration
- **Let** : conversion of value (null check)
- **Run** : Execute lambda with side-effects and no result
- **With** : Configure object created somewhere else

*Be careful when using these functions to avoid potential problems:*

- Do not use **with** on nullable variables.
- Avoid nesting **apply**, **run** and **with** as you will not know what is the current **this**.
- For nested **apply** and **let**, use named parameters instead of **it** for the same reason
- Avoid **it** in long call chains as it is not clear what it represents.

# Cheat Sheet - Introduction

```
class MyClass {  
  fun test() {  
    val str: String = "..."  
  
    val result = str.let {  
      print(this)  
      print(it)  
      42  
    }  
  }  
}
```

## **This (Receiver):**

This is an instance of MyClass (this@MyClass) while test() a method of MyClass is.

## **It (Argument):**

It is the String “...” on which we executed **let**.

## **42 (Return Value):**

42 is the result which will be returned from the block

| Function | Receiver (this) | Argument (It) | Result  |
|----------|-----------------|---------------|---------|
| Let      | This@MyClass    | String(“...”) | Int(42) |

# Cheat Sheet – for the Std.kt

| Funktion | Receiver (this) | Argument (It) | Result        |
|----------|-----------------|---------------|---------------|
| Let      | This@MyClass    | String("...") | Int(42)       |
| run      | String("...")   | n/a           | Int(42)       |
| run*     | this@MyClass    | n/a           | Int(42)       |
| with*    | String(„...“)   | n/a           | Int(42)       |
| apply    | String(„...“)   | n/a           | String(„...“) |
| also     | this@MyClass    | String(„...“) | String(„...“) |

*\* = No extension function. These methods have to be called in the old way*

End of section: Std. library  
Any questions ?